Assignment 04 - Hough Transforms

This assignment may be viewed in the original markdown format or as a pdf (`assignment04.pdf`) which renders the equations in a readable form. If you're curious, the pdf was created using a tool called `pandoc` which "compiles" from markdown to LaTeX then from LaTeX to pdf.

Problem 1: Points and lines in numpy (12 points)

Fill out the missing code in `points_and_lines.py` according to the specification in the type signature and docstring of each function. There are 6 functions, each worth 2 points. Note that some code to check the shapes of arrays is already provided. All functions are implementable with numpy vectorized calls (no python loops). Solutions that pass the tests but use python loops will receive 1 point per function.

To aid you, we've provided a unit test file `test_points_and_lines.py` that you can run to check your work. You can run it in the terminal with the command:

```
python -m unittest test_points_and_lines.py
```

Or by creating a "Unittests" run configuration in PyCharm that points to the `test_points_and_lines.py` file.

Note: during grading, we might run your code with additional unit tests not provided in `test_points_and_lines.py`. In other words, passing the unit tests in `test_points_and_lines.py` is not a guarantee that you'll get full credit. You'll need to think about other potential edge cases yourself. (You can safely assume that we will not try to "trick" your code by passing in arrays of the wrong shape or data type).

Problem 2: Using the Hough transform to find all the horizontal lines (15 points)

If you run the script `hough_lines.py` with the argument `house.jpg`, you should see that a new file `output_images/house_lines.jpg`is generated. Your task for Problem 1 is to figure out how to modify the code in `hough_lines.py` so that only the *horizontal* lines in the house image are detected and annotated. We'll define a horizontal line to be any line whose absolute value angle relative to the horizontal axis of the image is less than some configurable threshold, say 5 degrees (this is already a parameter in the code as `args.horizontal_window_degrees` but you need to uncomment it and implement code that uses it).

You are free to change other aspects of the code or use other values for any other parameters defined by `argparse`. For instance, you could do some color space transform to take advantage of the fact that the house is blue rather than just using `cv.COLOR_BGR2GRAY`. You may modify the `HoughLineDetector` class in whatever ways are necessary. The goal is to detect just the dominant horizontal lines such as the shadows cast by the siding of the house. In order to make this work, you'll need to understand how the `HoughLineDetector` class works, but ultimately you should only need to modify roughly 10 lines of the file.

All parameters should be configurable from the command line. For instance,

```
python hough_lines.py house.jpg --horizontal-window-degrees=2.0
```

should run and detect fewer lines than

```
python hough_lines.py house.jpg --horizontal-window-degrees=10.0
```

If we set a big enough range of angles like `--horizontal-window-degrees=90.0`, then all lines should be detected.

Do the following:

1. Make any modifications you need to `hough_lines.py` so that the `--horizontal-window-degrees` flag works as described above.
2. Figure out what other parameter settings make it work well. Do you need to modify the edge detector? Change thresholds? Change the window size for the non-maximal suppression? Change the accumulator array size?

3. Write a command-line command that we can run to reproduce your results:

   ```
   python hough_lines.py house.jpg YOUR VALUES FOR COMMAND-LINE ARGUMENTS HERE
   ```

4. Run it and ensure that `output_images/house_lines.jpg` contains the expected output, with just horizontal lines detected.

5. Summarize your approach in a few sentences. How did you go about it? What did you need to change and why?

   Answer: _____

Problem 3: Find the coins (15 points)

In assignment 2 you helped Mario find some coins using template matching. Now, let's detect real-world coins using Hough transforms. In this problem, you'll implement a few key missing methods in a Hough transform class that detects *circles*. The code for this is in `hough_circles.py`.

We'll constrain the problem a bit so that the user has to pass in the radius of the circles to be detected. This simplifies the problem because with only one radius to consider, we can use a 2D accumulator array instead of a 3D one. The accumulator array in `HoughCircleDetector` is stored in the `self.accumulator` attribute, which is a 2D numpy array. The value of `self.accumulator[i, j]` is the number of votes for the circle center at row `self.center_y[i]` and column `self.center_x[j]`. Beware the "flip" in convention from `(x, y)` to `(i, j)`, where `i` indexes `y` and `j` indexes `x`!

Before writing any code, see if you can write an equation for the shortest distance from an arbitrary point (x, y) to any point on a circle centered at (cx, cy) with radius r. Note: this is **not** the distance to the circle's center, but the distance to the circle itself. For instance, the distance from the point `(2, 1)` to a circle centered at the origin with radius 1 is 1.0, since the closest point on the circle is `(1, 0)`. The distance from the point `(0, 0.5)` to this same circle is `0.5` because the closest point on the circle is `(0, 0.5)`. You don't need to submit this equation, but you'll need to implement it below. **Hint:** you never actually need to know the actual point on the circle that is closest to `(x, y)`!

Now, see if you can implement this function using vectorized numpy calls. In other words, if I provide you with a numpy array called `centers_xy` of shape `(num_circles, 2)`, a point `xy` of shape `(2,)`, and a radius `r`, can you compute the shortest distance from `xy` to each of the `num_circles` circles in `centers_xy`? The result should be a numpy array of shape `(num_circles,)` containing the distances from `xy` to each circle (again, to the circle itself, not to the center of each circle). If you can write this using vectorized numpy calls (no python loops), it will make your Hough transform code much much faster.

Let's say you called your function `distance_to_circles`, and let's say that we call it like

distances = distance_to_circles(centers_xy, xy, r)

Then, the number of effective votes in the accumulator array for each circle should be 1.0 for any circles where `distances` is close to 0.0, and the number of votes should be smaller for circles where `distances` is larger. A sensible way to achieve this is by defining the number of votes as follows:

$$votes = exp(-distances^2/(2 * \sigma^2))$$

where `sigma` is a configurable parameter that is already initialized to a sensible value for you in the code. This is known as a "soft" voting scheme, since the votes are not binary (0 or 1) but rather a continuous value between 0 and 1.

Do the following:

1. Implement the `distance_to_circles` function described above in `hough_circles.py`.

2. Implement `HoughCircleDetector.add_edge_at_xy`. This method should add votes to the accumulator array using the above formulae, and should make use of the `distance_to_circles` function you implemented above.

3. Implement `HoughCircleDetector.get_circles` according to the specification in its docstring. This method should return a numpy array where rows are `[cx, cy]`, one for each circle detected. Use `HoughLineDetector.get_lines` as a reference. Be careful about the order of x and y coordinates! The convention is that points are (x,y), but note that numpy arrays are indexed as `arr[i, j]` where `i` is an index on the y-axis and `j` is an index on the x-axis.

4. Run the script `hough_circles.py` with the argument `coins.jpg`. You should see a new file `output_images/coins_circles30.jpg` generated. If you run it with the argument `coins.jpg --radius=20`, you should see a new file `output_images/coins_circles_radius20.jpg` generated.

   Notice that the coins.jpg image shows a pile of US coins occluding each other. Let's see if we can find good command-line arguments for `hough_circles.py` to detect each type of coin (each is a different radius). The default parameters are not ideal. You should play around with a few values for the different parameters available via argparse, such as the parameters that are fed to the Canny edge detector. A good guess for the radius of Quarters in this image is about 55 pixels. Pennies have a radius of about 45 pixels. Dimes are about 40. Nickels are about 48.

   Create the best command-line command you can to detect each type of coin. For instance, if you think you can detect Quarters with a radius of 55 pixels, you might run:

   ```
   python hough_circles.py coins.jpg --radius=55 OTHER ARGUMENTS HERE
   ```

   Write your commands below, and include the output images in your submission. You should have one image for each type of coin, and each image should show the detected circles for that coin type.

   Command for detecting quarters: _____

   Command for detecting pennies: _____

   Command for detecting dimes: _____

   Command for detecting nickels: _____

5. Summarize your results. Were some coins harder to detect than others? What do you think caused false positives (detecting coins where there aren't any)? What about false negatives (failing to detect coins that are there)? Did there tend to be more false positives or false negatives in certain parts of the image?

   Answer: _____

Collaboration and Generative AI disclosure

Did you collaborate with anyone? Did you use any Generative AI tools? Briefly explain what you did here.

Reflection on learning objectives

This is optional to disclose, but it helps us improve the course if you can give feedback.

- what did you take away from this assignment?
- what did you spend the most time on?
- about how much time did you spend total?
- what was easier or harder than expected?
- how could class time have better prepared you for this assignment?

Submitting

Your submission should consist of a single zip file named like `firstname_lastname_assignment4.zip`. The zip file should contain all the contents of this directory, including this file (which you should have written some answers into above) and any images or other outputs generated by your code.